Concurrency
pick the low-hanging fruit

Václav Pech

LET'S MOVE THE JAVA WORLD

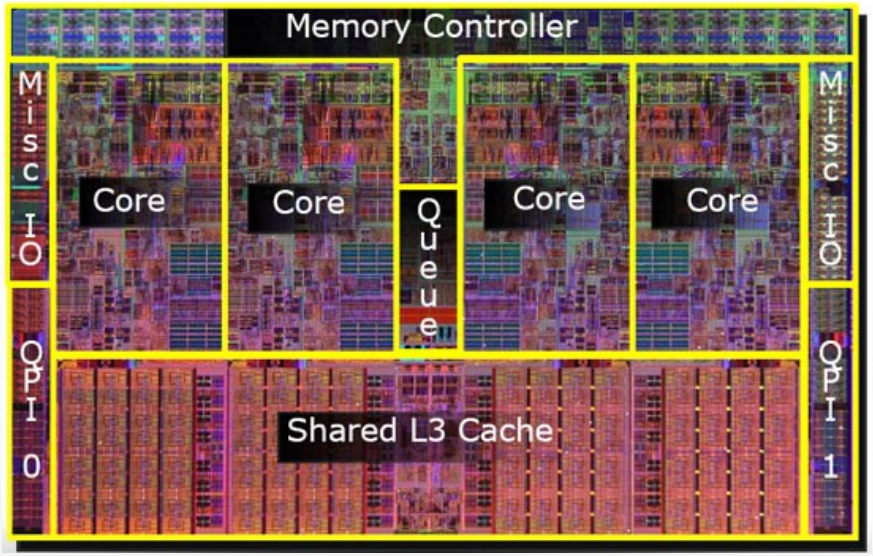
GeeCON

About me

Passionate programmer
Concurrency enthusiast

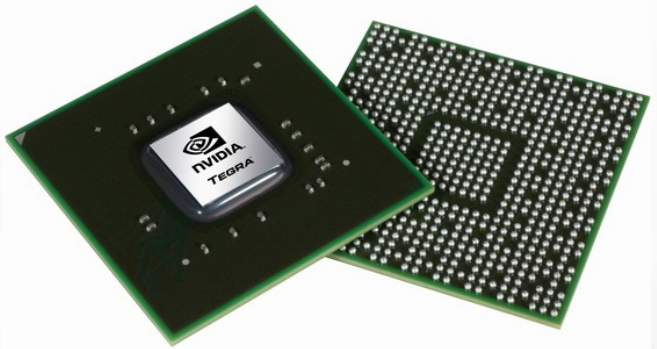
GPars @ Codehaus lead
Groovy contributor
Technology evangelist @ JetBrains
JetBrains Academy member

http://www.jroller.com/vaclav
http://twitter.com/vaclav_pech



We’re all in the parallel computing business!


LG
5:56 PM
London
12/13 2010 MON
05
56
Browser
Gmail
Market
Camera
Phone
Contacts
Messaging
Applicati...


NVIDIA
TEGRA

of cores

Dealing with threads sucks!

```
public class Counter {
    private static long count = 0;

    public Counter() {

        count++;

    }
}
```

Dealing with threads sucks!

```
public class Counter {
    private static long count = 0;

    public Counter() {
        synchronized (this) {
            count++;
        }
    }
}
```

Dealing with threads sucks!

```
public class Counter {
    private static long count = 0;

    public Counter() {
        synchronized (this.getClass()) {
            count++;
        }
    }
}
```

Dealing with threads sucks!

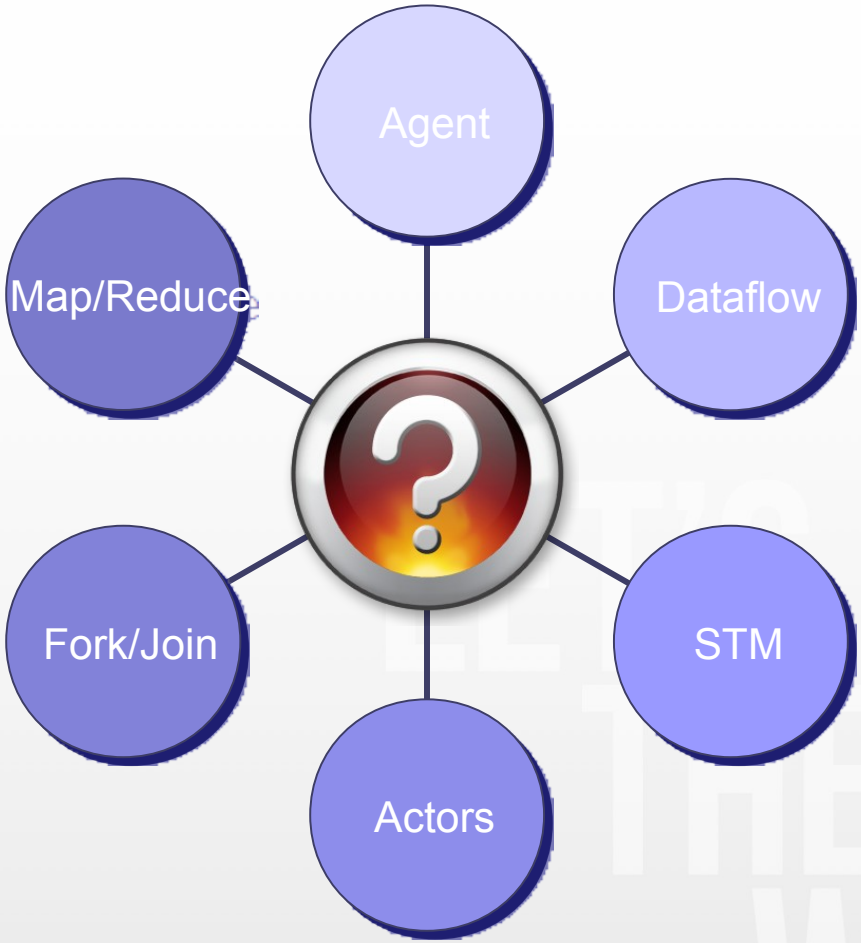
```
public class ClickCounter implements ActionListener {
    public ClickCounter(JButton button) {
        button.addActionListener(this);
    }

    public void actionPerformed(final ActionEvent e) {
        ...
    }
}
```

Dealing with threads sucks!

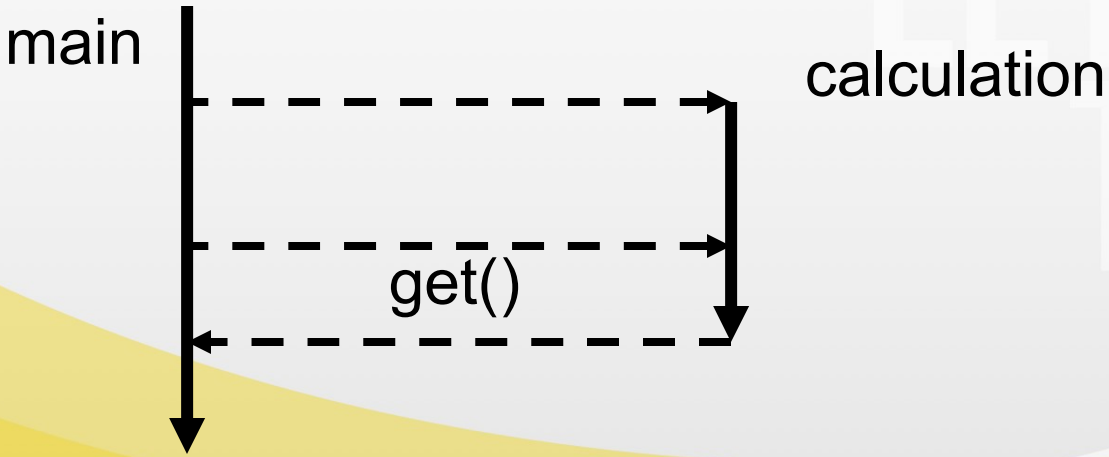
Dead-locks, Live-locks, Race conditions, Starvation, ...

Can we do better?

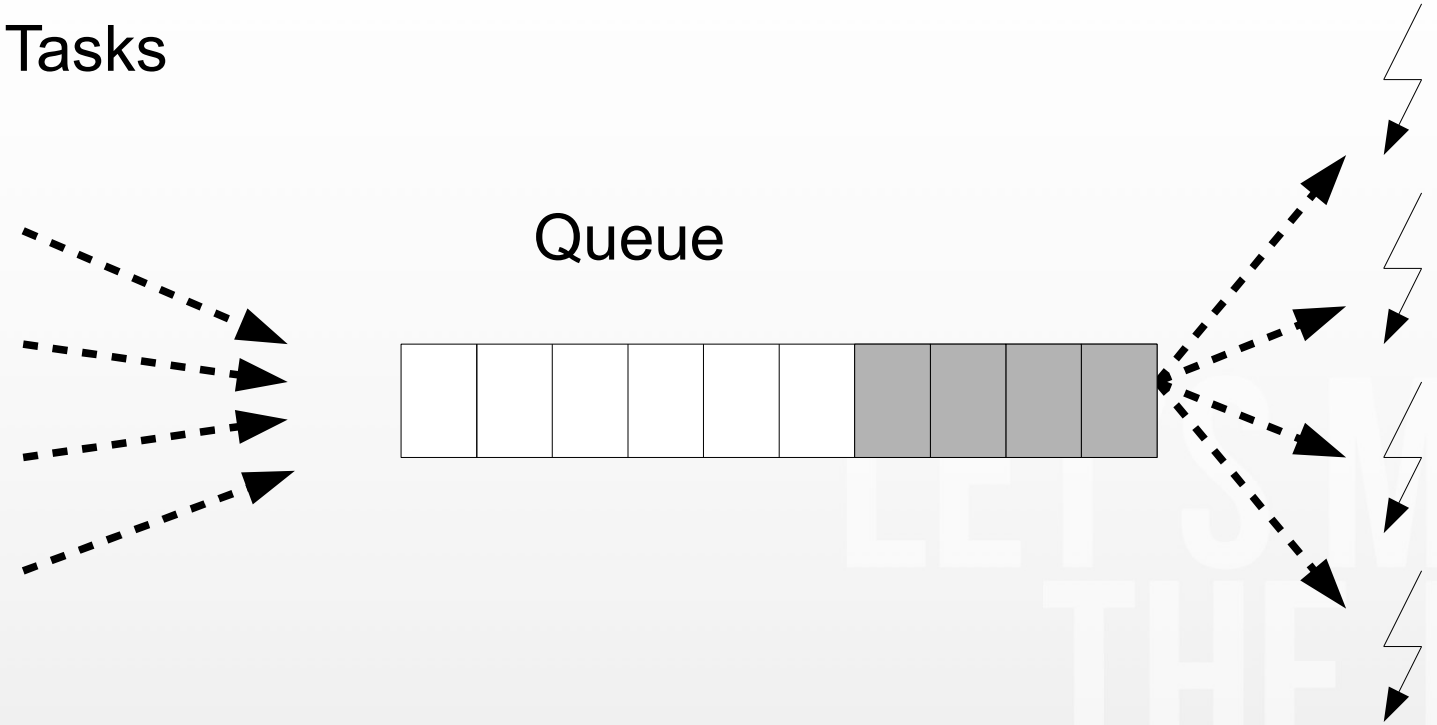


Asynchronous invocation

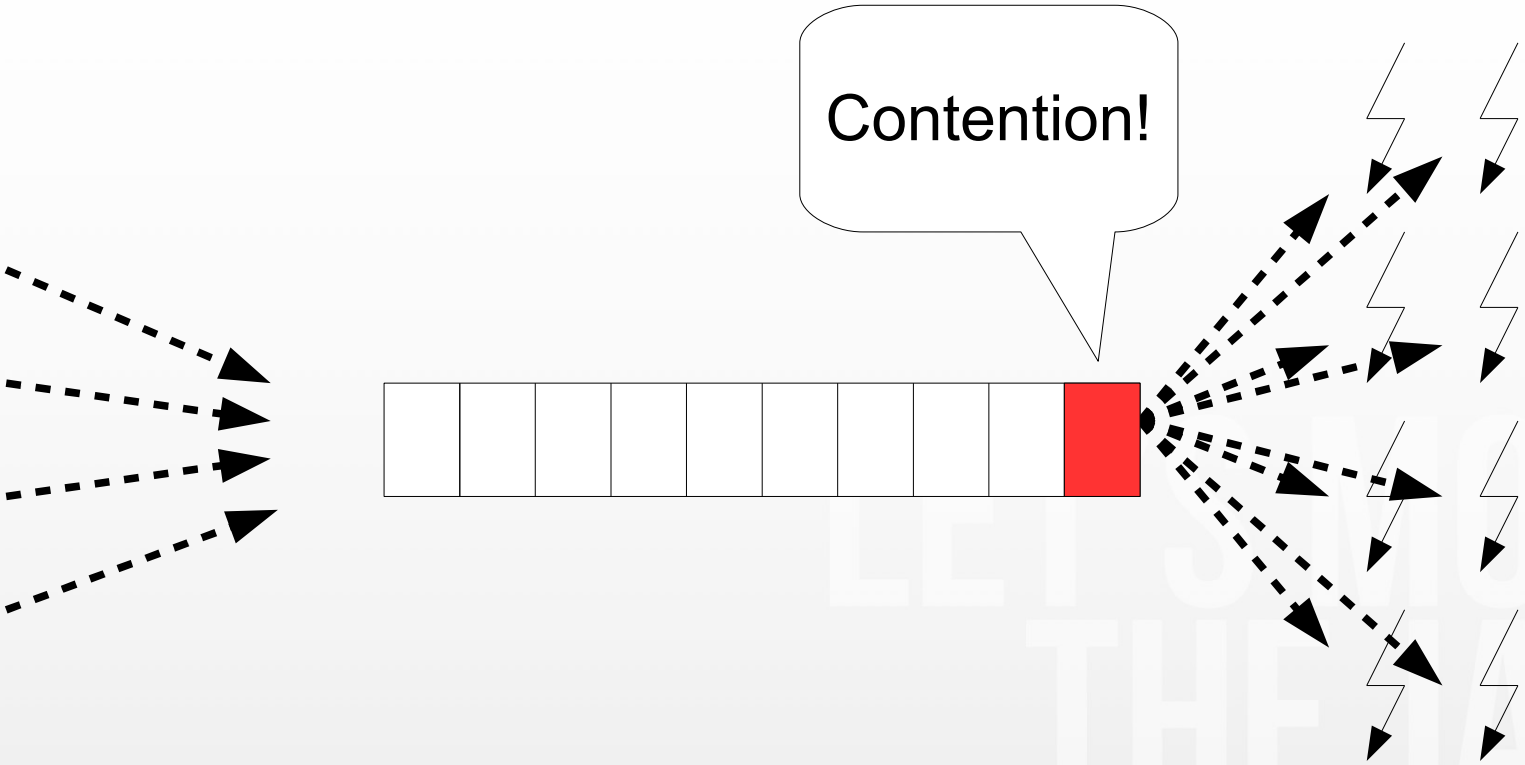
```
Future f = threadPool.submit(calculation);
…
System.out.println(“Result: “ + f.get());
```



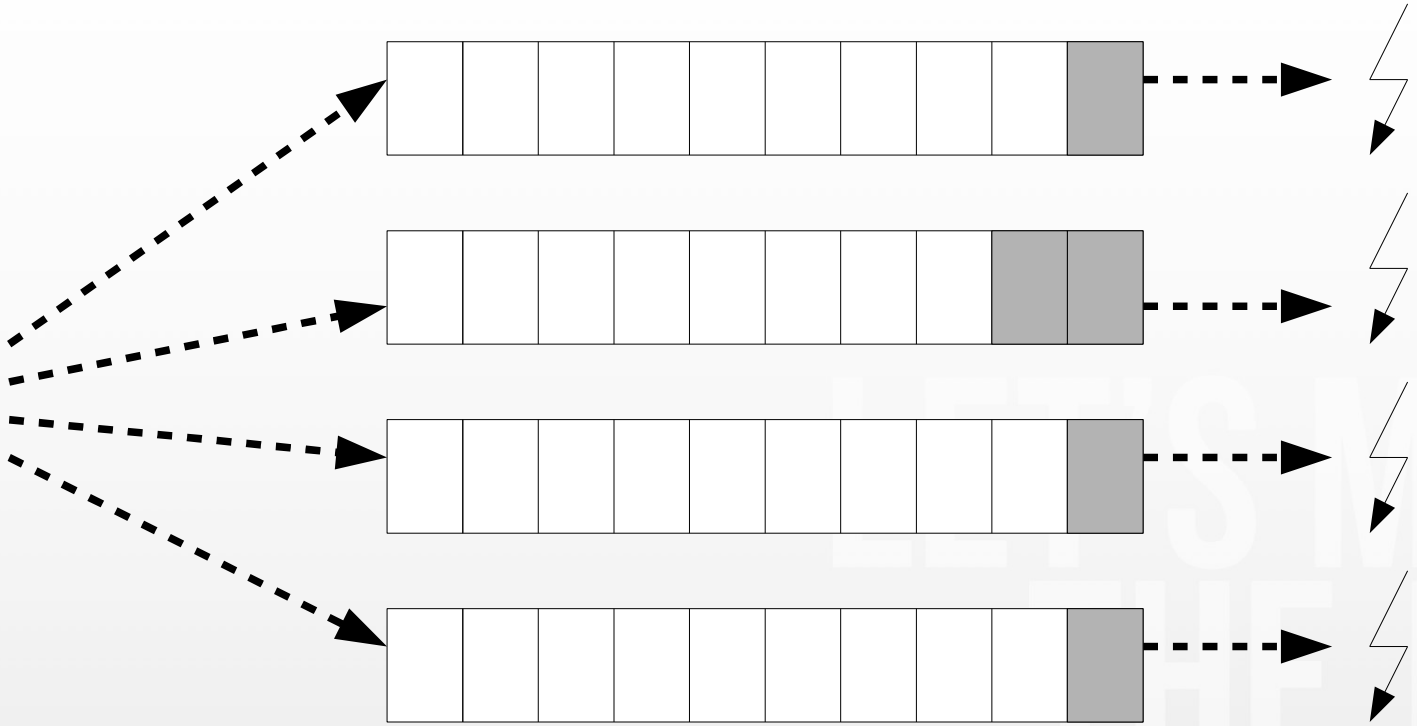
Thread Pool



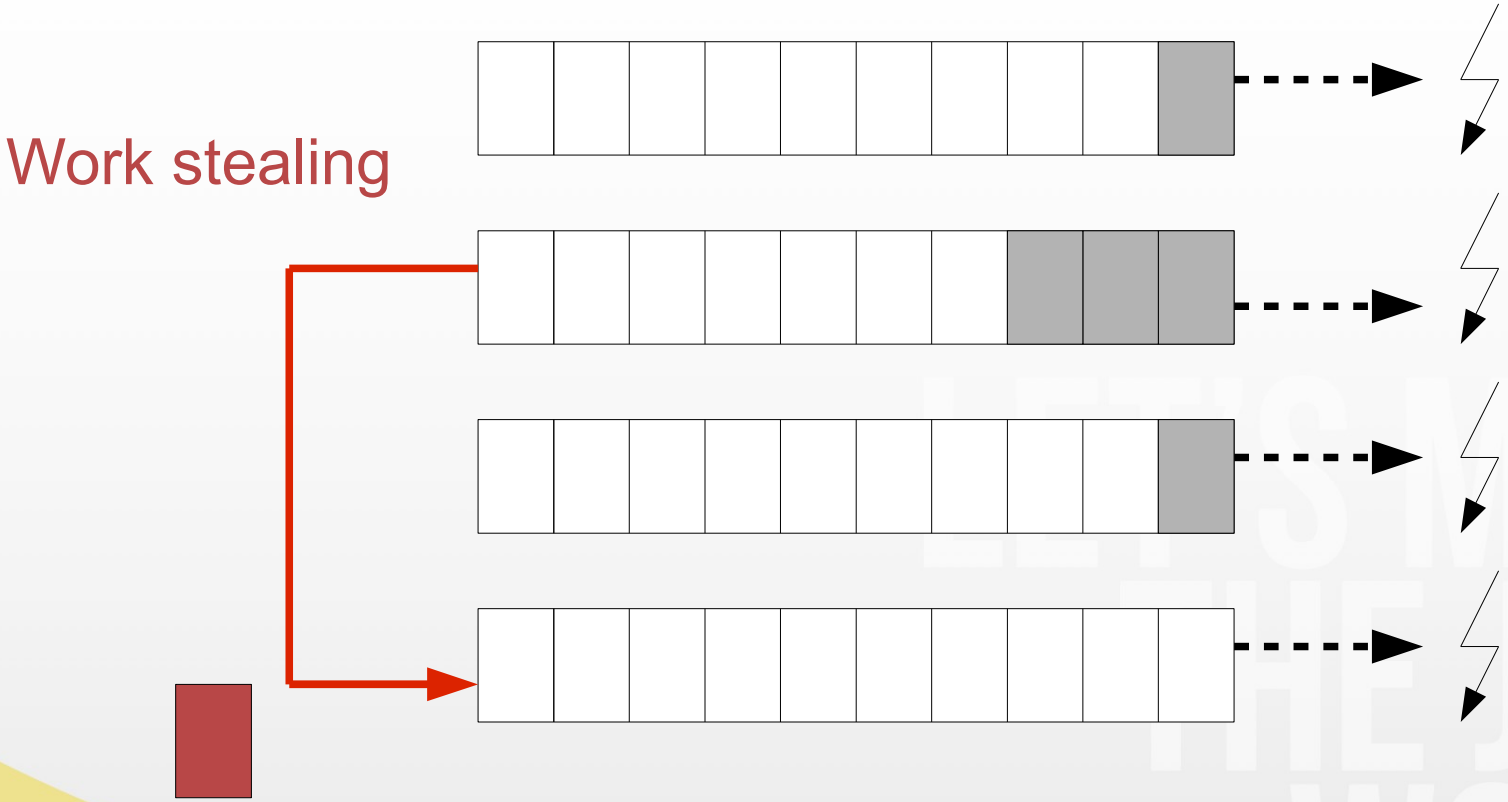
Thread Pool



Fork/Join Thread Pool

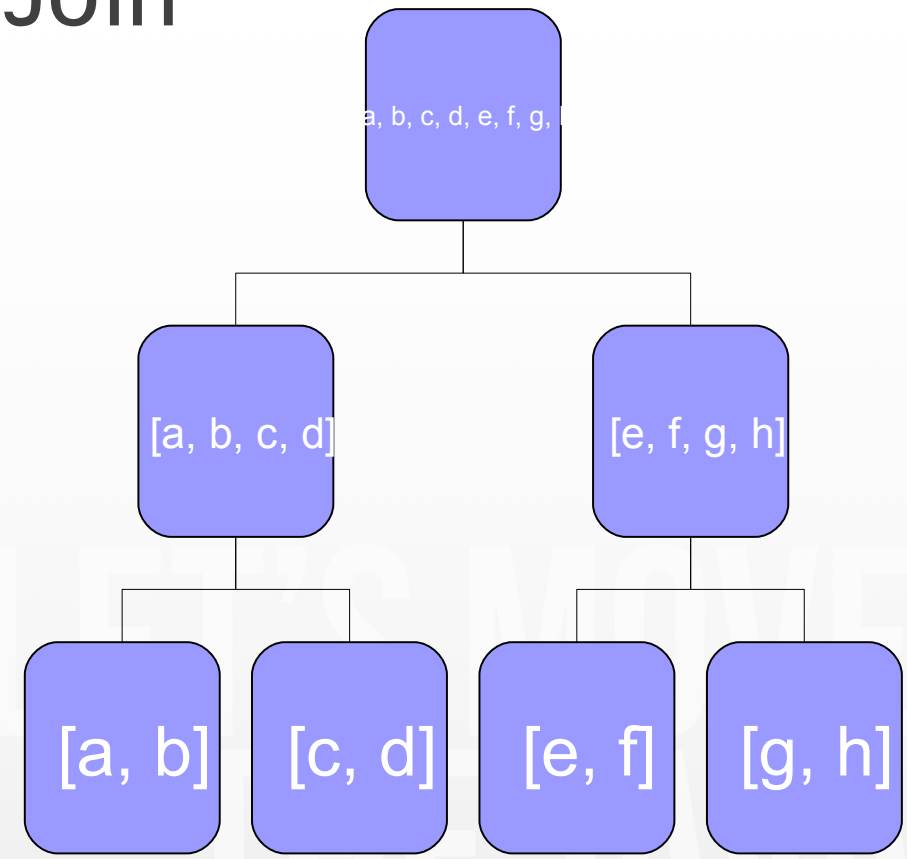


Fork/Join Thread Pool



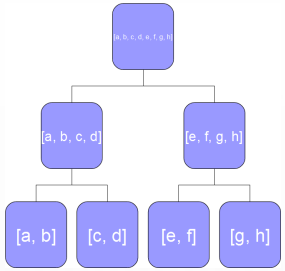
Fork/Join

- Solve hierarchical problems
  - Divide and conquer
  - Merge sort, Quick sort
  - Tree traversal
  - File scan / search
  - …



Collections (Groovy/GPars)

```
images.eachParallel {it.process()}
```




```
documents.sumParallel()
```

```
candidates.maxParallel {it.salary}.marry()
```

Parallel Arrays (jsr-166y)

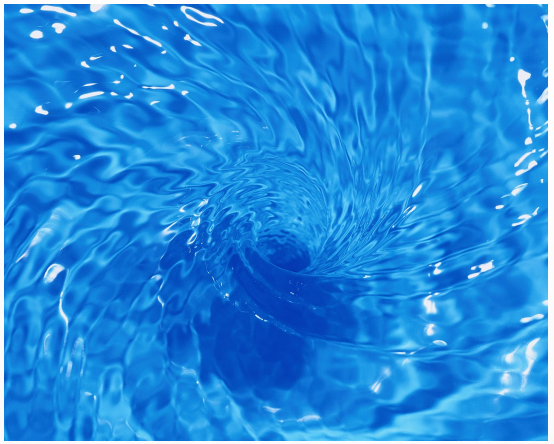
```
ParallelArray namesOfWomen =

    people.withFilter(aWoman).withMapping(retrieveName).all()
;
```

```
Ops.Predicate aWoman = new Ops.Predicate() {
  public boolean op(Person friend) {return !friend.isMale();}
};

Ops.Op retrieveName = new Ops.Op() {
  public Object op(Person friend) {return friend.getName();}
};
```

Dataflow Concurrency



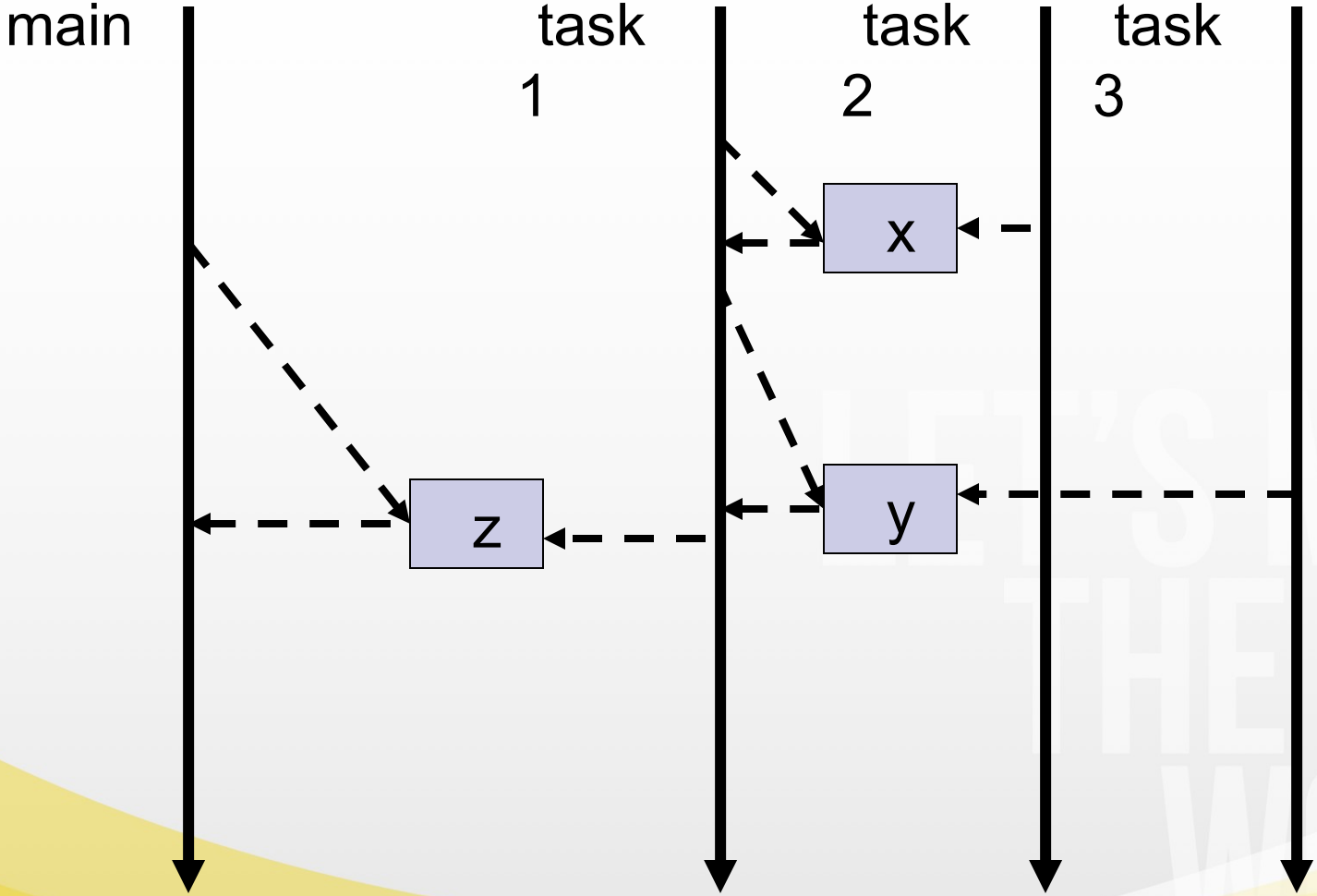
- No race-conditions
- No live-locks
- Deterministic deadlocks

  Completely deterministic programs

BEAUTIFUL code

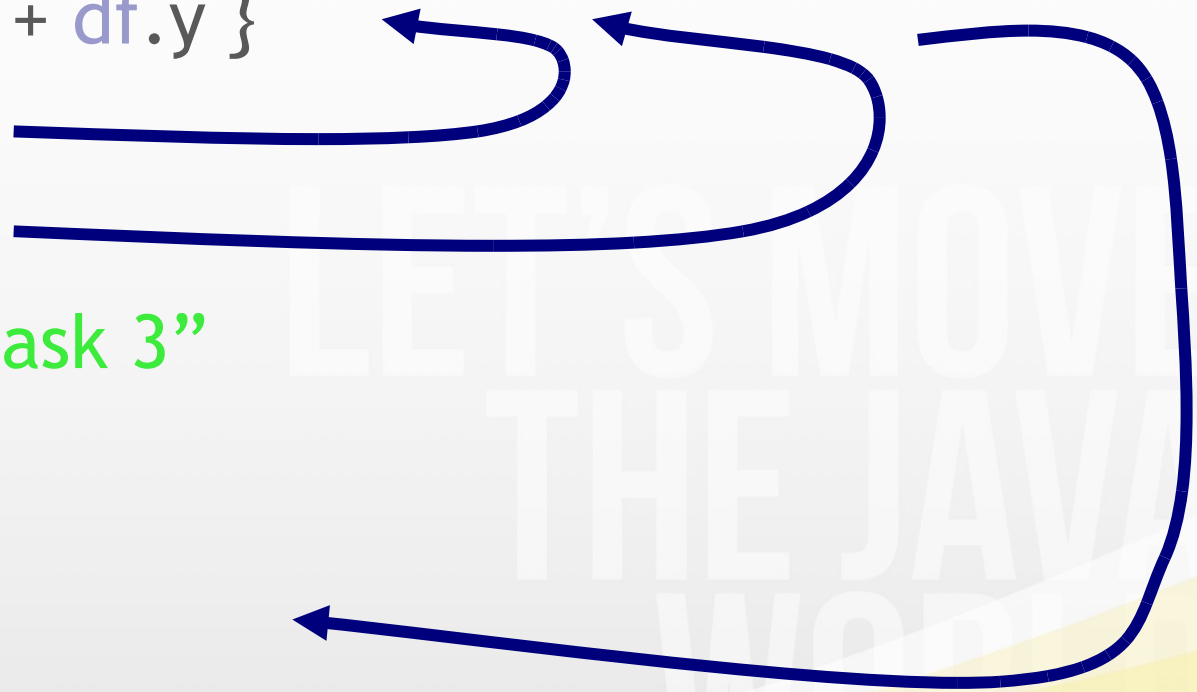
(Jonas Bonér)

Dataflow Variables / Promises



DataFlows (GPars)

```
def df = new DataFlows()
task { df.z = df.x + df.y }
task { df.x = 10 }
task {
    println ”I am task 3”
    df.y = 5
}
assert 15 == df.z
```



Composing async functions

```
int hash1 = hash(download('http://www.gpars.org'))
int hash2 = hash(loadFile('/gpars/website/index.html'))
boolean result = compare(hash1, hash2)
println result
```

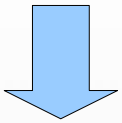
Composing async functions

```
def hash = oldHash.asyncFun()
def compare = oldCompare.asyncFun()
def download = oldDownload.asyncFun()
def loadFile = oldLoadFile.asyncFun()

def hash1 = hash(download('http://www.gpars.org'))
def hash2 = hash(loadFile('/gpars/website/index.html'))
def result = compare(hash1, hash2)
println result.get()
```

Composing async functions

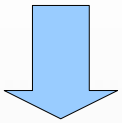
int hash(String text) {…}



Promise<int> hash(*Promise<String>* | *String* text)

Composing async functions

int hash(String text) {…}



Promise<int> hash(*Promise<String>* | *String* text) {

1. Return a Promise for the result
2. Wait (non-blocking) for the text param
3. Call the original *hash()*
4. Bind the result

}

Composing async functions

Combine functions as usual

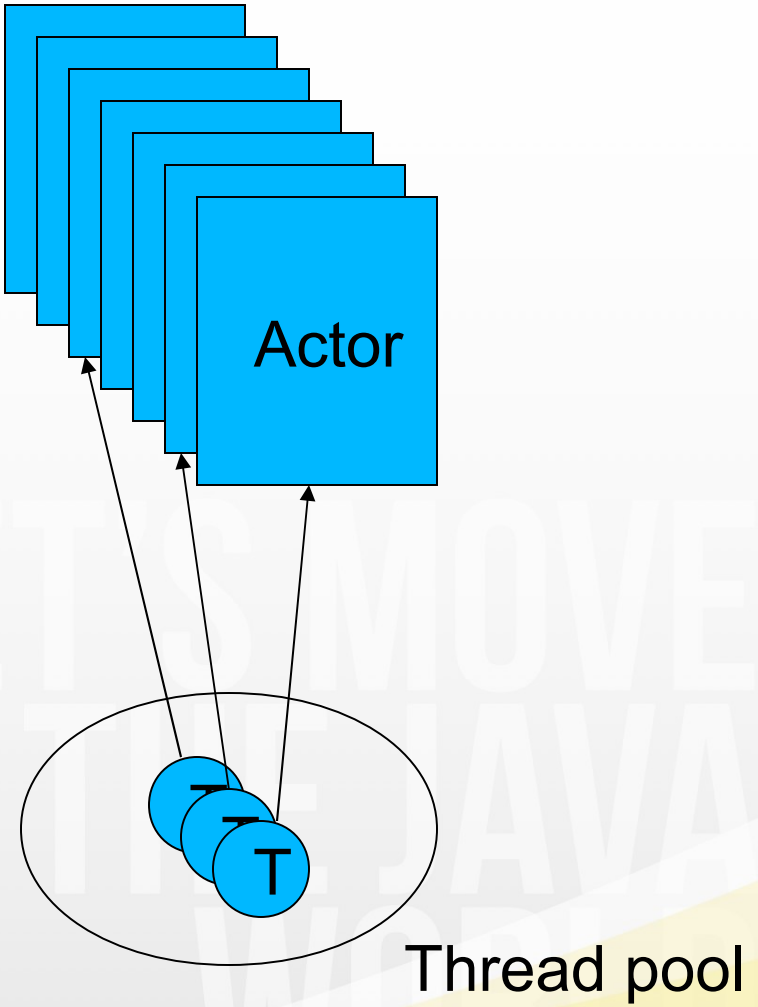
Parallelism is detected automatically

Milestone

- Asynchronous calculations
- Fork/Join
- Parallel collection processing
- Dataflow variables/streams

Actors

- Isolated
- Communicating
  - Immutable messages
- Active
  - Pooled shared threads
- Activities
  - Create a new actor
  - Send a message
  - Receive a message



Stateless Actors (pure Java)

```
class MyActor extends DynamicDispatchActor {
    private Account account = ...
    public void onMessage(String msg) {
        String encripted = encrypt(msg);
        reply(encripted);
    }
    public void onMessage(Integer number) {
        reply(2 * number);
    }
    public void onMessage(Money cash) {
        System.out.println("Received a donation " + cash);
        account.deposit(cash);
    }
}
```

Active Objects

```
@ActiveObject
class MyCounter {
    private int counter = 0

 @ActiveMethod
    def incrementBy(int value) {
        println "Received an integer: $value"
        this.counter += value
    }
}
```

No more threads and locks

```
images.eachParallel {
    //concurrency agnostic code here
}

def myActor = actor {
    //concurrency agnostic code here
}

atomic { /*concurrency agnostic code here*/ }
…
```

Summary

Parallelism is not hard, multi-threading is

Jon Kerridge, Napier University

Questions?

Find more at:

- http://gpars.codehaus.org
- http://www.jroller.com/vaclav
- http://twitter.com/vaclav_pech